ESP8266

Interface GPIO

Version 0.5

Espressif Systems IOT Team

http://bbs.espressif.com

Table of Contents

1. Overview

The ESP8266 has 16 general IOs. Their pin numbers and names are shown in the table below:

GPIO NO.	pin NO.	pin name
GPIO0	pin15	GPIO0_U
GPIO1	pin26	U0TXD_U
GPIO2	pin14	GPIO2_U
GPIO3	pin25	U0RXD_U
GPIO4	pin16	GPIO4_U
GPIO5	pin24	GPIO5_U
GPIO6	pin21	SD_CLK_U
GPIO7	pin22	SD_DATA0_U
GPIO8	pin23	SD_DATA1_U
GPIO9	pin18	SD_DATA2_U
GPIO10	pin19	SD_DATA3_U
GPIO11	pin20	SD_CMD_U
GPIO12	pin10	MTDI_U
GPIO13	pin12	MTCK_U
GPIO14	pin9	MTMS_U
GPIO15	pin13	MTDO_U

In the QUAD mode flash, 6 IO interfaces are used for flash communication.

In the DUAL mode flash, 4 IO interfaces are used for flash communication.

Users may find the following documents helpful:

List of GPIO registers: ***ESP8266_GPIO_Register.xlsx***

List of ESP8266 pin functions: ***ESP8266_Pin_List.xlsx***

2. Instruction on GPIO registers

2.1. GPIO function selection register

The ESP8266 MTDI is used to demonstrate the GPIO function selection.

Function selection register `PERIPHS_IO_MUX_MTDI_U` (this register differs for different GPIOs)

`PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTDI_U,FUNC_GPIO12);`

`FUNC_GPIO12=3`.

Configurations differ for different pins.

During the configuration, refer to ***ESP8266_Pin_List_xxxxxx.xlsx***. On the ***Digital Die Pin List*** page, users can see the general GPIO and their multiple functions. On the ***Reg*** page, users can find registers related to GPIO function selection.

On the ***Digital Die Pin List*** page, users can find the function configuration in the FUNCTION column.

Notice: if you want to configure it to be FUNCTION X, write X -1 into the bit in the register. For example, if you want to configure it to be FUNCTION 3, write 2 into the bit in the register.

2.2. GPIO output registers

2.2.1. Output enable register: GPIO_ENABLE_W1TS

bit[15:0] the output enable bit (readable and writable):

If the related bit is set to be 1, the IO output is enabled. Bit[15:0] contains 16 GPIO output enable bits.

2.2.2. Output disable register: GPIO_ENABLE_W1TC

bit[15:0] the output disable bit (readable and writable):

If the related bit is set to be 1, the IO output is disabled. Bit[15:0] contains 16 GPIO output disable bits.

2.2.3. Output enable status register: GPIO_ENABLE

bit[15:0] the output enable status bit (readable and writable):

Value of bit[15:0] of this register shows the related pin output enable status.

By writing data into bit[15:0] of `GPIO_ENABLE_W1TS` and bit[15:0] of `GPIO_ENABLE_W1TC`, users can control bit[15:0] of `GPIO_ENABLE`. For example, when bit[0] of `GPIO_ENABLE_W1T` is set to be 1, then bit[0] of `GPIO_ENABLE` =1; when bit[1] of `GPIO_ENABLE_W1TC` is set to be 1, then bit[1] of `GPIO_ENABLE` = 0.

2.2.4. Output low level register GPIO_OUT_W1TC

bit[15:0] output low level bit (write only register):

If the related bit is set to be 1, the IO output is low level (at the same time, users should enable the output). Bit[15:0] contains 16 GPIO output statuses.

Note: if users need to set the pin to high level, they need to configure the `GPIO_OUT_W1T` register.

2.2.5. Output high level register GPIO_OUT_W1TS

bit[15:0] output high level bit (write only register):

If the related bit is set to be 1, it means the IO output is high level (at the same time, users should enable the output). Bit[15:0] contains 16 GPIO output statuses.

Notice: if users need to set the pin to low level, they need to configure the `GPIO_OUT_W1TC` register.

2.2.6. Output status register GPIO_OUT

bit[15:0] output status bit (read/write register):

Value of bit[15:0] of this register shows the related pin output status.

Bit[15:0] of `GPIO_OUT` is decided by bit[15:0] of `GPIO_OUT_W1TS` and bit[15:0] of `GPIO_OUT_W1TC`. For example, when bit[1] of `GPIO_OUT_W1TS` =1, then `GPIO_OUT`[1] =1; when bit[2] of `GPIO_OUT_W1TC` = 1, then `GPIO_OUT`[2]=0.

2.3. GPIO input register

2.3.1. Input status register GPIO_IN

bit[15:0] the input status bit (readable and writable):

If the related bit is set to be 1, the IO pin status is high level. If the related bit is set to be 0, the IO pin status is low level. Bit[15:0] contains 16 GPIO input status bits.

Notice: the GPIO input detection function is enabled by default.

2.4. GPIO interrupt registers

2.4.1. Interrupt type register GPIO_PIN12 (this register differs for different GPIOs)

bit[9:7] (readable and writable):

0: the GPIO interrupt is disabled

1: rising edge triggered interrupt

2: falling edge triggered interrupt

3: double-edge triggered interrupt

4: low level

5: high level

2.4.2. Interrupt status register GPIO_STATUS

Bit[15:0] (readable and writable):

If the related bit is set to be 1, the IO interrupts. Bit[15:0] contains 16 GPIOs.

2.4.3. Interrupt clearing register GPIO_STATUS_W1TC

Bit[15:0] (readable and writable):

Write 1 into the related bit, the related GPIO interrupt status will be cleared.

2.5. GPIO16 related APIs

Different from other IO interfaces, GPIO16(`XPD_DCDC`) belongs to the RTC module instead of the general GPIO module. It can be used to wake up the chip during deep-sleep; it can be configured to input or output mode; but it cannot trigger the IO interrupt. the APIs are shown below.

2.5.1. gpio16_output_conf(void)

Set the GPIO16 to the output mode.

2.5.2. gpio16_output_set(uint8 value)

Output high/low level from GPIO16. Configure GPIO16 to the output mode first.

2.5.3. gpio16_input_conf(void)

Set the GPIO16 to the input mode.

2.5.4. gpio16_input_get(void)

Read the GPIO16 input level status. Configure GPIO16 to the input mode first.

3. Parameter configuration

Three scenes are given as examples for parameter configuration:

1. Configure the MTDI output high level, and enable the pull up.
2. Configure the MTDI to the input mode, and get its level status.
3. Configure the MTDI to falling edge triggers interrupt.

3.1. Parameter configuration for Scene 1

- Step 1: configure the MTDI to GPIO mode.

  ```
  PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTDI_U,FUNC_GPIO12);
  ```

  This sentence writes 1 into bits 4-5 of `PERIPHS_IO_MUX_MTDI_U` register. When bits 4-5 of `PERIPHS_IO_MUX_MTDI_U` are set to be 1, the MTDI is configured to the GPIO mode. For details of `PERIPHS_IO_MUX_MTDI_U` register, refer to ***Section 2, Instruction on GPIO register***.

- Step 2: configure the MTDI output high level

  ```
  GPIO_OUTPUT_SET(GPIO_ID_PIN(12), 1);
  ```

  This sentence has two functions:

  Write 1 into bit 12 of `GPIO_ENABLE_W1TS` register. It enables the MTDI output function.

  Write 1 into bit 12 of `GPIO_OUT_W1TS` register. It sets MTDI output to high level.

  Notice:

  To set MTDI output to low level, set the second parameter of this function to be 0.

  ```
  GPIO_OUTPUT_SET(GPIO_ID_PIN(12), 0);
  ```

  This sentence has two functions:

  Write 1 into bit 12 of `GPIO_ENABLE_W1TS` register. It enables the MTDI output function.

  Write 1 into bit 12 of `GPIO_OUT_W1TC` register. It sets MTDI output to low level.

- Step 3: enable the MTDI pull up.

  ```
  PIN_PULLUP_EN(PERIPHS_IO_MUX_MTDI_U);
  ```

  It writes 1 into bit 7 of `PERIPHS_IO_MUX_MTDI_U`. It enables the MTDI pull up.

  Notice: to disable the MTDI pull up, use the following sentence:

  ```
  PIN_PULLUP_DIS(PERIPHS_IO_MUX_MTDI_U);
  ```

3.2. Parameter configuration for Scene 2

- Step 1: configure the MTDI to GPIO mode.

```
PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTDI_U,FUNC_GPIO12);
```

This sentence writes 1 into bits 4-5 of `PERIPHS_IO_MUX_MTDI_U` register. When bits 4-5 of `PERIPHS_IO_MUX_MTDI_U` are set to be 1, the MTDI is configured to the GPIO mode.

- Step 2: configure the MTDI to the input mode.

```
GPIO_DIS_OUTPUT(GPIO_ID_PIN(12));
```

- Step 3: get the MTDI pin level status.

```
Uint8 level=0;
level=GPIO_INPUT_GET(GPIO_ID_PIN(12));
```

`GPIO_INPUT_GET(GPIO_ID_PIN(12))` gets the status of bit 12 of `GPIO_IN` register. The value of this register shows the input level of related pin. (Enable the input function of the related pin first to get effective register status)

Notices:

If MTDI is at high level, then the return value of `GPIO_INPUT_GET` is 1, level = 1;

If MTDI is at low level, then the return value of `GPIO_INPUT_GET` is 0, level = 0.

3.3. Parameter configuration for Scene 3

```
typedef enum {
GPIO_PIN_INTR_DISABLE = 0,
GPIO_PIN_INTR_POSEDGE = 1,
GPIO_PIN_INTR_NEGEDGE = 2,
GPIO_PIN_INTR_ANYEGDE = 3,
GPIO_PIN_INTR_LOLEVEL = 4,
GPIO_PIN_INTR_HILEVEL = 5
} GPIO_INT_TYPE;
```

This structure is used to configure the GPIO interrupt trigger manner. It is declared in `gpio.h`.

- Step 1: configure the MTDI to GPIO mode.

```
PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTDI_U,FUNC_GPIO12);
```

This sentence writes 1 into bits 4-5 of `PERIPHS_IO_MUX_MTDI_U` register. When bits 4-5 of `PERIPHS_IO_MUX_MTDI_U` are set to be 1, the MTDI is configured to the GPIO mode.

- Step 2: configure the MTDI to the input mode.

```
GPIO_DIS_OUTPUT(GPIO_ID_PIN(12));
```

- Step 3: disable all IO interrupts.

```
ETS_GPIO_INTR_DISABLE();
```

- Step 4: set the interrupt handler function.

```
ETS_GPIO_INTR_ATTACH(GPIO_INTERRUPT,NULL);
```

- Step 5: configure MTDI to falling edge triggers interrupt.

  `gpio_pin_intr_state_set(GPIO_ID_PIN(12),GPIO_PIN_INTR_NEGEDGE);`

  This sentence writes 0x02 into bit[9:7] of `GPIO_PIN12` register. It sets MTDI to falling edge triggers interrupt.

  Notice: If users want to disable the MTDI interrupt function , write 0x02 into bit[9:7] of `GPIO_PIN12` register. For other interrupt triggering mode configuration, refer to ***Section 2, Instruction on GPIO register***.

- Step6: enable the GPIO interrupt.

  `ETS_GPIO_INTR_ENABLE();`

3.4. Interrupt function processing procedures

- Step 1: clear the interrupt.

  ```
  Uint16 gpio_status=0;
  gpio_status = GPIO_REG_READ(GPIO_STATUS_ADDRESS);
  GPIO_REG_WRITE(GPIO_STATUS_W1TC_ADDRESS, gpio_status);
  ```

  For instructions on `GPIO_STATUS` and `GPIO_STATUS_W1TC`, refer to ***Section 2, Instruction on GPIO register***.

- Step 2: check which IO triggered the interrupt (when multiple IOs are configured to be in interrupt mode)

  `If(gpio_status==GPIO_Pin_12)`

- Step 3: if it is double-edge triggered interrupt, check whether this interrupt is triggered by rising or falling edge.

  `if(!GPIO_INPUT_GET(GPIO_ID_PIN(12)))` // if this MTDI interrupt is triggered by falling edge.

3.5. Example of the interrupt function processing procedures

```
void gpio_intr_handler()
{

    uint32 gpio_status = GPIO_REG_READ(GPIO_STATUS_ADDRESS);//read interrupt status
    uint8  level=0;
     GPIO_REG_WRITE(GPIO_STATUS_W1TC_ADDRESS, gpio_status);//clear interrupt mask
     if(gpio_status & (BIT(12))){  //judge whether interrupt source is gpio12
       if(GPIO_INPUT_GET(12)){   // if gpio 12 is high level

         }else{ // if gpio 12 is low level

      }
    }
     else{
     }

}
```